A Fault-Tolerant Approach to Secure Information Retrieval

Erica Y. Yang, Jie Xu and Keith H. Bennett
Department of Computer Science
University of Durham,
Durham DH1 3LE, UK
{Erica.Yang, Jie.Xu, Keith.Bennett}@durham.ac.uk

Abstract

Several Private Information Retrieval (PIR) schemes were proposed to protect users' privacy when sensitive information stored in database servers is retrieved. However, existing PIR schemes assume that any attack to the servers does not change the information stored and any computational results. We present a novel fault-tolerant PIR scheme (called FT-PIR) that protects users' privacy and at the same time ensures service availability in the presence of malicious server faults. Our scheme neither relies on any unproven cryptographic assumptions nor the availability of tamper-proof hardware. A probabilistic verification function is introduced into the scheme to detect corrupted results.

Unlike previous PIR research that attempted mainly to demonstrate the theoretical feasibility of PIR, we have actually implemented both a PIR scheme and our FT-PIR scheme in a distributed database environment. The experimental and analytical results show that only modest performance overhead is introduced by FT-PIR while comparing with PIR in the fault-free cases. The FT-PIR scheme tolerates a variety of server faults effectively. In certain fail-stop fault scenarios, FT-PIR performs even better than PIR. It was observed that 35.82% less processing time was actually needed for FT-PIR to tolerate one server fault.

Keywords: Distributed database systems, fault tolerance, malicious faults, private information retrieval, secret sharing, security

1. Introduction

Critical information services such as online stock information systems and medical records databases are increasingly becoming accessible through the Internet. Such applications give rise to significant security and reliability problems. From the viewpoint of a system user, there are at least two fundamental requirements: privacy protection and assurance of service availability. While attempting to meet these requirements, a system has to cope with software bugs, operator mistakes, and malicious attacks – the common causes of service interruption [6].

Protecting the privacy of a user is concerned with a method for protecting *the identity of the information* the user is interested in (i.e. the *intention*) against any attacks occurring on the information system side. For example, when querying an online stock information system, an investor is usually reluctant to reveal the specific stock of interest to any other parties including the operators who manage the system. This problem is traditionally called Private Information Retrieval (PIR).

Chor et al [7] identified the PIR problem in 1995 and proposed several PIR schemes. Their schemes used a so-called honest-but-curious model in which an attacker may observe the transactions performed between a user and the information system (so as to find out the intention of the user) but *not* act maliciously. Most research in this field [1, 3, 4, 7, 8, 14, 15] has used the same assumption for potential attacks.

A PIR scheme may require a set of replicated servers (RS) or use only a single server (SS). In the RS-based approach [1, 3, 7, 8], identical database servers are replicated on separate nodes of a distributed system. It is assumed that communication between the replicated servers is limited so that the servers cannot collude together in trying to violate the user's privacy [7]. It follows that, given unlimited resources, any individual server cannot gain any information of the user's intention. In the SS-based approach, no replication of servers is needed but the computational capability of servers must be bounded. For example, the work reported in [4, 14, 15] assumes the difficulty of breaking some number-theoretical problems (e.g. the existence of one-way functions). As long as these problems are still hard to solve in a computational sense, the user's privacy is preserved.

The existing PIR solutions do not tolerate any type of server faults. The honest-but-curious model they used is unrealistic. In practice, an attacker may actually spy on transactions occurring between a user and a system, stop

running the servers, tamper with the information stored in the servers, and maliciously manipulate results returned to the user. A more realistic fault model is urgently needed. The previous PIR work has also focussed on demonstrating the theoretical possibility of PIR without any practical implementation and experimentation attempted. The binary bit string model used by the existing work is too restricted to be utilised in real systems.

In this paper we introduce a *fault-tolerant* approach to secure information retrieval that guarantees both users' privacy and service availability even in the presence of malicious server faults. We present the design and implementation details of a fault-tolerant PIR scheme. Our work also demonstrates the practical feasibility of applying both PIR and FT-PIR to a realistic database system and provides significant experimental results and performance analysis.

The rest of this paper is organised as follows. Section 2 introduces the system model and gives an overview of FT-PIR. Section 3 presents the formal description of FT-PIR and discusses a verification function for detecting corrupted results. Section 4 gives the design and implementation details of a PIR scheme and the FT-PIR scheme. Section 5 presents experiments and performance analysis, and Section 6 discusses the related work briefly. Section 7 concludes the paper.

2. System Model

Consider a synchronous distributed system with a set of processing nodes connected by a communication network. It is assumed that any message passing amongst the nodes is bounded within a given period of time. The system contains a set of k replicated servers (called replicas) $\{S_1, S_2, \ldots, S_k\}$ and a set of clients, running on separated processing nodes. The replicas provide information services to the clients and the clients take inputs from the users of the system. A replica can simultaneously deal with multiple clients' queries, while a client may need to send queries to multiple replicas to utilise a service.

A replica stores information using a database or a file system. The information is identical for all replicas and is modelled as a character string $x = x_1 x_2 \ldots x_n$ of length n. Each character x_j, where $j \in \{1, 2, \ldots, n\}$, is an integer taken from a given integer set $[X\text{-}1] = \{0, 1, \ldots, X\text{-}1\}$. For example, if we encode characters using the ASCII code, an eight-bit byte character can be viewed as an integer taken from the set $\{0, 1, \ldots, 255\}$. In fact the range of this set can be adjusted dynamically according to the actual data status.

In order to perform certain operations on $x \in [X\text{-}1]^n$, we associate the set $[X\text{-}1]$ with a finite field $GF(q)$, where q is a prime number and $q \geq X$. Let $[q\text{-}1] = \{0, 1, 2, \ldots, q\text{-}1\}$ be the set of q elements of $GF(q)$. We have $x \in [X\text{-}1]^n \subseteq [q\text{-}1]^n$. Note that the results of operations over a finite field, e.g. addition, multiplication, subtraction, and division, are still elements of the finite field [16].

Protection of Privacy: We now explain briefly the principle of private information retrieval. Suppose that a user is interested in the character x_i stored in a system. Within the system, a client takes i as the input from the user, where $i \in \{1, 2, \ldots, n\}$. In order to keep this user's intention private from any replica, the client constructs k *query functions* $Q_1, \ldots, Q_k$, based on i and some random inputs and generates a set of *random and thereby independent* queries. These queries will then be sent to the replicas respectively.

There are k *answer functions* $A_1, \ldots, A_k$ that are defined for the replicas respectively and perform read-only operations on x. Based on the query submitted from the client, each replica executes its answer function to generate an answer and sends the answer back to the client. The client will then reconstruct x_i locally by executing a *reconstruction function*. Finally, the correct value of x_i will be passed to the user.

Fault Model: In this paper we take server faults into account only. The replicated servers in a system can be malicious and behave arbitrarily. The information stored, queries and answers may be corrupted. A server may return a spurious answer, or collude with other malicious servers to violate users' privacy.

Fault Tolerance: In order to tolerate malicious server faults, redundant but different queries have to be generated by the client. Based on a threshold number of correct answers from those fault-free replicas, the client will be able to reconstruct a correct result. To reduce the number of redundant queries needed, a verification function may be used to validate the result.

Our system model differs from the PIR model developed in [7] in several ways. First, the model in [7] uses a *binary* bit string to model the information that is hard to implement in practice. Our character string model is much more realistic and has been implemented in an actual setting. Secondly, we add the fault tolerance capability to PIR by a combined use of redundancy and result verification. Thirdly, our model assumes malicious servers while the model in [7] says that servers may be curious but always honest.

3. An FT-PIR Scheme

Consider k replicated servers. Let t be the maximum number of faulty replicas, r be a random input to the query functions, and L_r be the length of the random input. L_q is the length of a query, L_a is the length of an answer, and $s_1, \ldots, s_t, s_{t+1}$ are elements taken from the set $\{1, 2, \ldots, k\}$.

The FT-PIR Scheme: A (t, k) FT-PIR scheme is a message passing scheme, where $k \geq 2t + 1$, and consists of

k query functions Q_1, ..., Q_k:

$$\{1, 2, \ldots, n\} \times [q\text{-}1]^{Lr} \mapsto [q\text{-}1]^{Lq}$$

k answer functions A_1, ..., A_k:

$$[X\text{-}1]^n \times [q\text{-}1]^{Lq} \mapsto [q\text{-}1]^{La}$$

A reconstruction function R:

$$\{1, 2, \ldots, n\} \times [q\text{-}1]^{Lr} \times ([q\text{-}1]^{La})^{t+1} \mapsto [q\text{-}1]$$

A verification function V:

A result *res* is valid if and only if

res is the intended result $x_i \in [X\text{-}1]$

The scheme should satisfy the following properties:

a) Correctness: For every $x \in [X\text{-}1]^n$, $i \in \{1, 2, \ldots, n\}$, and $r \in [q\text{-}1]^{Lr}$, $\exists\ s_1, \ldots, s_t, s_{t+1} \in \{1, 2, \ldots, k\}$, $R(i, r, A_{s1}(x, Q_{s1}(i, r)), \ldots, A_{st+1}(x, Q_{st+1}(i, r))) = x_i$.

b) Privacy: For every $i, j \in \{1, 2, \ldots, n\}$, every $s_1, \ldots, s_t \in \{1, 2, \ldots, k\}$, and $Q \in [q\text{-}1]^{Lq}$,

$$\mathbf{Pr}\ ((Q_{s1}(i, r), Q_{s2}(i, r), \ldots, Q_{st}(i, r)) = Q) = \mathbf{Pr}\ ((Q_{s1}(j, r), Q_{s2}(j, r), \ldots, Q_{st}(j, r)) = Q)$$

where the probabilities **Pr**'s are taken over uniformly and randomly chosen $r \in [q\text{-}1]^{Lr}$.

c) Safety: The scheme will output only the intended result $x_i \in [X\text{-}1]$.

d) Liveness: The scheme eventually terminates.

Property a) states that there exists at least one set of replicas (i.e. availability) whose answers can be used to reconstruct the intended result (i.e. correctness). Property b) means that from any set of t queries, it is impossible to decide which specific data item the user is interested in since the joint distribution of $Q_{s1}(i, r)$, $Q_{s2}(i, r)$, ..., $Q_{st}(i, r)$ is independent of i. In other words, from t or less queries it is theoretically impossible for a replica or a group of replicas to gain any information about i. Property c) is guaranteed by the verification function. Property d) holds provided that $k \geq 2t + 1$.

3.1 Construction of an FT-PIR Scheme

A PIR scheme was developed in [7] using the method of low degree polynomial interpolation [2]. An FT-PIR scheme can be constructed based on the PIR scheme, with additional queries. For every $j \in \{1, 2, \ldots, n\}$, we define a function j:$[n] \mapsto \{0, 1\}$, so that for every $l = 1, 2, \ldots, n$, $j(l) = 1$, if $l = j$, otherwise $j(l) = 0$. Each query function consists of n degree-t polynomials:

$$g_l(z) \equiv i(l) + r_{l1}*z + r_{l2}*z^2 + \ldots + r_{lt}*z^t \quad \text{for } l = 1, 2, \ldots, n$$

where the free term of the l-th polynomial is $i(l)$ and $r_{l1}, r_{l2, \ldots,} r_{lt} \in GF(q)$.

The client needs to select k non-zero distinct points from $GF(q)$, denoted by $m_1, m_2, \ldots, m_k$ and then sends the tuple $<g_1(m_d), g_2(m_d), \ldots, g_n(m_d)>$ as a query to replica S_d for $d = 1, 2, \ldots, k$.

Each answer function will be in the form:

$$F^{i,x}(z) = \sum_{j=1}^{n} g_j(z) * x_j$$

By the constructions, we know that

Property 1: $F^{i, x}(z)$ is a polynomial (in z) of degree at most t.

Property 2: The free term of $F^{i, x}(z)$ is $F^{i, x}(0) = x_i$.

S_d will send the value of $F^{i, x}(m_d)$ back to the client. From k replicas, the client will obtain the values of the polynomial $F^{i, x}(z)$ at $2t + 1$ distinct points. Based on any $t + 1$ value pairs, the client will be able to use the Lagrange Interpolation formula to reconstruct a result. Without loss of generality, assume that the $t + 1$ value pairs are $(m_1, F^{i, x}(m_1))$, $(m_2, F^{i, x}(m_2))$, …, $(m_{t+1}, F^{i, x}(m_{t+1}))$. The reconstruction function $F^{i, x}(z)$ can be defined as follows.

$$F^{i,x}(z) = \sum_{j=1}^{t+1} F^{i,x}(m_j) \prod_{\substack{p=1 \\ p \neq j}}^{t+1} \frac{z - m_p}{m_j - m_p}$$

3.2 Probabilistic Result Verification

It is important to notice that the condition $k \geq 2t + 1$ implies only the existence of a correct result. In order to identify the correct result as the system output, we have to either design a perfect verification function or introduce further redundancy into queries. Both solutions are costly. Probabilistic result verification is desirable in this case since it keeps additional redundancy at an acceptable level.

The polynomial interpolation-based PIR schemes and our FT-PIR scheme essentially share the same spirit of Shamir's secret sharing scheme [18]. All these schemes exploit the polynomial properties (i.e. perfect secrecy and interpolation uniqueness [12]) for providing privacy protection and fault-tolerant operations. It is therefore possible to apply the existing results of secret sharing to both PIR and FT-PIR schemes. In the following, we develop a probabilistic verification function for FT-PIR, based on Tompa and Woll's modification [19] on the Shamir's secret sharing scheme. This function does not guarantee perfect result verification but detects corrupted results with a high (and adjustable) probability.

The main idea of the probabilistic verification function is to limit the valid range of reconstructed results. Because a character x_j ($j \in \{1, 2, \ldots, n\}$) is an integer taken from a pre-known set $[X\text{-}1]$, for every x_j, there are exactly X candidates of *valid results*. Note that all calculations and functions are performed over the finite field $GF(q)$. There will be q possible reconstructed *results* for x_j over the set $[q\text{-}1]$. It follows immediately that if a reconstructed result is within $[X\text{-}1]$, it is valid. Otherwise, it is invalid.

Recall that $[X\text{-}1] \subseteq [q\text{-}1]$. We can increase the size of $GF(q)$ such that most of corrupted results will appear in the set $[q\text{-}1] - [X\text{-}1]$. In other words, the probability of undetected errors can be confined within a pre-defined bound e for any $e > 0$. Although it is possible that a valid result in $[X\text{-}1]$ is in fact a corrupted result (i.e. not the intended result), the probability that this actually happens can be controlled by adjusting the field size intentionally.

According to [19], for a (t, k) FT-PIR scheme, if $q > \max[(X - 1)(t/e) + t + 1, k]$, the probability of undetected errors will be less than the arbitrarily small e (for the proof, readers are referred to [19]). This ensures that q is large enough to reveal the corrupted results with a high probability $(1– e)$, e.g. 99.99%. Further discussion about how to decide the q in our implemented system will be detailed in Section 4 and 5.

This verification function does not rely on any unproven cryptographic premise (e.g. intractability of factorisation of big primes) and on the availability tamper-proof hardware (e.g. secure processors). The probabilistic assurance is guaranteed unconditionally provided that the q is determined by the above formula.

3.3 A Control Algorithm for FT-PIR

We will describe a control algorithm that we used to implement the FT-PIR scheme. It is assumed that a *RESULT* value is valid if and only if $RESULT \in [X\text{-}1]$. The control algorithm first initialises the *RESULT* to be –1 (a dummy value and viewed as an impossible result). The algorithm accepts only a valid *RESULT* and rejects any invalid results. The `groupCounter` variable counts the number of candidate result groups checked so far.

1. $RESULT = -1$.
2. A user feeds an index i to the system.
3. Use the query functions to generate k queries.
4. Wait until at least $t + 1$ answers available.
5. Set `groupCounter` to 1;
6. If `groupCounter` $> \binom{k}{t+1}$, stop.

   (In this case, there must be more than t faulty servers and the scheme will not be able to reconstruct the result.)
7. Select a new group of $t + 1$ candidate answers.
8. Based on the answers in the group, execute the reconstruction function to get a *RESULT*.
9. Perform the verification function: if the *RESULT* is valid, go to step 10. Otherwise, increment `groupCounter` and go to step 6.
10. Output the *RESULT* to the user and stop.

We can now prove that the FT-PIR scheme with probabilistic result verification satisfies the properties specified in Section 2, provided that $k \geq 2t + 1$.

Theorem 1: The FT-PIR scheme satisfies the correctness property.

Sketch of Proof: By the Lagrange Interpolation theorem [16], $t + 1$ distinct points are necessary and sufficient to uniquely determine a polynomial over a finite field of degree at most t. By **Property 2**, the free term of the polynomial is uniquely determined. Hence, in the FT-PIR scheme, $t + 1$ correct answers can uniquely and correctly determine the free term of polynomial $F^{i, x}(z)$, i.e. x_i.

Q.E.D.

Theorem 2: The FT-PIR scheme satisfies the privacy property.

Sketch of Proof: By the Lagrange Interpolation theorem, we know that for a degree t polynomial, the knowledge of t (or fewer) points indicates no information about the polynomial, i.e. no information about its free term. Since the FT-PIR scheme uses the Lagrange Interpolation formula as the reconstruction function, any t or fewer points indicates no information about the free term $F^{i, x}(0)$, i.e. x_i.

Q.E.D.

The scheme will only satisfy the safety property with a pre-defined probability, but guarantee the liveness property.

4. Implementation

This section describes implementations of three information retrieval schemes: a normal scheme called NIR, the polynomial interpolation PIR scheme described in [7], and our FT-PIR scheme. The NIR scheme retrieves database records by sending SQL queries in a normal way. For NIR there is no protection for users' privacy and no fault tolerance. The PIR scheme protects users' privacy only against honest-but-curious servers, while the FT-PIR scheme guarantees both users' privacy and service availability. An identical system architecture, depicted in Figure 1, was used for all implementations to ensure a fair comparison basis.

The communication between the client and replicated servers are implemented using Java sockets. The input from the user will be the index of specific record in the server database and the result will be the actual database record. The `cld` is a multithreaded client daemon. It interfaces with user-level applications and performs the following tasks: taking inputs from the user and returning results to it. The `cld` spawns k client threads $\texttt{cldt}_1$, $\texttt{cldt}_2$, …, $\texttt{cldt}_k$ to handle communications with servers concurrently. There are four major modules `init()`, `prepareQuery()`, `sendQuery()`, `performReconstruction()` in the `cld` daemon program for all the implementations. The module `init()` takes parameters from a configuration file to initialise the system. In the current implementation, this file resembles

the queries based on the user's input. The `prepareQuery()` module prepares the queries. The `sendQuery()` module creates *k* client threads and starts to run them immediately. After finishing the `sendQuery()` module, *k* client threads will communicate with *k* servers concurrently. The `performReconstruction()` module waits until there is sufficient number of answers available for the reconstruction operation. For the NIR scheme, there is no need to perform "reconstruction" since the results will be an actual record.

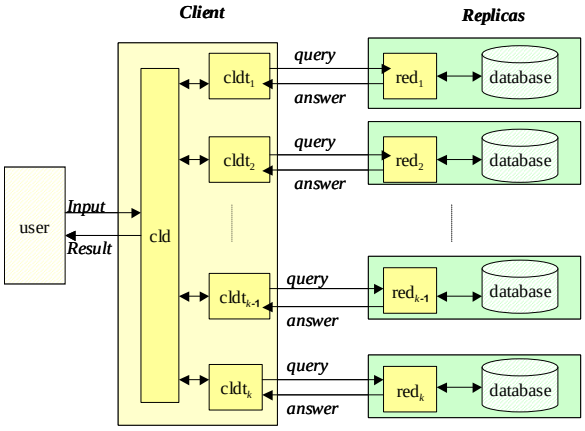

Figure 1. The system architecture.

In the NIR implementation, the configuration file contains only the information for the index of data item interested: `IndexOfDataItemIntersted`. A normal SQL statement will be generated by the `sendQuery()` module. The `performReconstruction()` module outputs the records returned from servers as the final result.

For PIR, the configuration file contains two pieces of extra information: `RandomSeed` and `MaximumNumberOfHonestButCuriousReplicas`. The `prepareQuery()` module generates queries by setting the order of the finite field and performing calculations over the field.

In the FT-PIR implementation, the configuration file contains information: `RangeOfData`, `MaximumNumberOfFaultyReplicas`, and `UndetectedErrorRate`. The `prepareQuery()` module does a job similar to PIR, but the calculations will take longer time due to the increased order of finite fields. The `performReconstruction()` module performs the reconstruction function. Instead of waiting for all answers to return, it only has to wait for a sufficient number (i.e. $t + 1$) of answers back from replicas to start the reconstruction process.

On the server side, the server daemon `red` is also a multithreaded program, concurrently dealing with multiple queries from clients. For clarity, we did not illustrate the multithread feature of `red` in Figure 1. The *k* replica daemons are denoted by $\texttt{red}_i$, where $i = 1, 2, \ldots, k$. When $\texttt{red}_i$ gets a client's query, it spawns a replica thread `redt` to deal with the query according to the answer function. In the current implementation, each server thread connects to a back-end database via the Java's JDBC driver.

For PIR and FT-PIR implantations, two types of messages needed to be passed between clients and servers: handshake messages and protocol messages. At the handshake stage, a client thread sends a message in the form of <handshake, nameOfDatabase> to the server daemon. Each replica thread will reply with a handshake message <handshake, lengthOfDatabase, lengthOfRecord>. The client thread will then start to send queries to servers.

Finally, two key modules `DBTransformation()` and `cal_Fp()` will be executed by the PIR and FT-PIR server threads. The former pre-processes the entire database and transforms it to an integer array. The latter uses the integer array and the query to produce an answer in the form of a string. Since these implementations do not modify any records and schemas stored in the replicated databases, we believe that the implementations described in this section are applicable to other realistic database or file systems.

5. Experiments and Analysis

This section describes the experiments and analysis based on our previous implementations. We will examine the performance overheads imposed by both PIR and FT-PIR, especially their relative performance in a fault-free setting as well as in a faulty environment.

5.1 Experimental Environment

The computers used in our experiments were connected via a 10MB/sec Ethernet LAN. The network delay, measured using the `ping` command, is always less than or equal to 10ms. The client machine is a SUN SPARC workstation running Solaris 8.0. All the server machines have the same specifications: 400 MHz Pentium IIs (celeron) running RedHat Linux (6.0 or 7.2), 3Com EtherLink XL 10Mb Ethernet NIC (3C900B-COMBO), 64 Mbytes of memory, and a 4 Gigabyte hard disk. The software used is: Sun JDK (JDK 1.3.1_03 mixed mode), MySQL 3.23, and MySQL JDBC Driver mm.mysql-2.0.4. Software was installed locally to ensure the independency of severs. Each server hosts an identical MySQL database containing 10 records. The size of each record is 110 bytes.

For PIR, all the computations were performed over the finite field *GF*(257). For FT-PIR, the range of data (`RangeOfData`) was set to be 255 since all the possible

characters are of the form of ASCII code. The value of `UndetectedErrorRate` was set to be 0.03, meaning that the verification function detects an invalid result with a probability 97%. The order of finite field can be adjusted intentionally. For example, when k = 3 (thus tolerating one server fault), the field is set to $GF(8501)$. When k = 11, the field is fixed to $GF(42349)$.

The garbage collection feature of Java often causes performance instability. To overcome this problem, all the performance data presented below are averages of more than 100 runs.

5.2 Experiments

We are mainly interested the average time taken to get a result for the user. As described in the following equation, the total processing time (TPT) consists of TPQ (time taken to prepare a query), TSQ (time taken to send a query), TWRG (time taken to wait for a ready answer group), and TR (time taken to reconstruct a result).

$$TPT = TPQ + TSQ + TWRG + TR.$$

We also are interested in the maximum time used for dealing with all the queries sent to the replicas, denoted by TDQmax. Since the client daemon interacts with replicas concurrently, TDQmax reflects the actual speed of replicas.

A. Comparison between PIR and FT-PIR in a Fault-Free Environment

We performed an initial performance comparison between NIR and various PIR schemes in [21]. The results showed that both PIR and FT-PIR generally double the execution cost of NIR. We now investigate the extra cost introduced by the FT-PIR scheme against the PIR scheme. This extra cost comes from the extra time used to compute over a bigger finite field, to generate more queries, to deal with more concurrent threads, and to verify results.

To start the reconstruction process, PIR needs all the $t + 1$ answers returned, while FT-PIR only needs to wait for the first group of $t + 1$ answers from the $k = 2t + 1$ servers returned. Therefore, FT-PIR will automatically use the answers from fast machines and ignore the slow machines.

The average of TPT for PIR is about 307ms and the standard deviation is always within 6%. As shown in Figure 2, the TPT of PIR did not increase as the number of replicas increased. This is because PIR's TR increased very slowly, as shown in Figure 3. It is worth to note that when k = 3 and 9, the TPT of PIR is longer than the others. This is possibly due to the fact that the actual performance of machines in this experiment varies quite a lot although they have the same specifications.

Also in Figure 2, FT-PIR displays a different picture. When $k \leq 9$, the TPT of FT-PIR is fairly stable. The TPT of FT-PIR's will not increase much even if the number of faults needed to be tolerated increases. As shown in Table 1, when the number of replicas was increased from 3 to 5 (i.e. from $k3t1$ to $k5t2$), only 3.6% extra processing time was imposed on the TPT. But FT-PIR's TPT increased quite dramatically when $k > 9$ because FT-PIR's TR started to become a major factor (58.91%) of TPT.

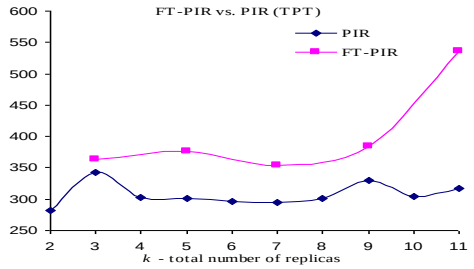

Figure 2. FT-PIR vs. PIR: total processing time (ms).

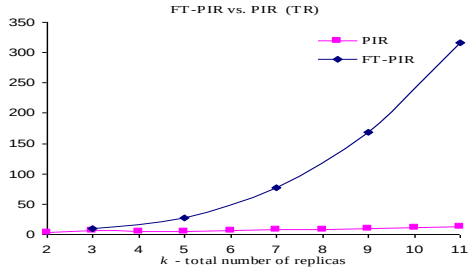

Figure 3. FT-PIR vs. PIR: reconstruction time (ms).

As shown in Table 1, the performance overhead introduced by FT-PIR is about 6% when t = 1 and k = 3, in comparison with the corresponding PIR implementation. In order to tolerate up to two faults, the FT-PIR imposes about 26% overhead. When the maximum number of faults does not exceed four, the performance overhead of FT-PIR is always less than 26%. This is a desirable feature because tolerating up to four faulty replicas could satisfy most real-world applications.

Figure 4 and 5 show the respective contribution of each specific factor to the total processing time. In PIR, the major performance costs are the waiting time (i.e. TWRG). The time spent on query preparation (TPQ) and sending query (TSQ) contributes a little to TPT for both schemes. TR increases steadily but it is still a minor contributor in the PIR implementation. In FT-PIR, both waiting time (i.e. TWRG) and TR are significant. In particular, if the number of elements (i.e. q) of the finite field increases, more time has to be spent on reconstructing a result. For example, when k = 3, q = 8501, and when k = 11, q = 42349. Apart from spending time on result reconstruction, FT-PIR spends extra time to verify the results.

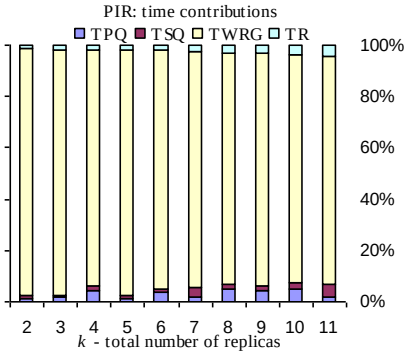

Figure 4. Time contributions of PIR.

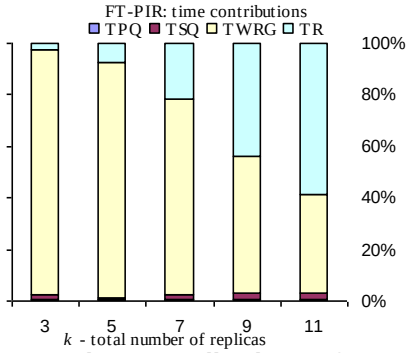

Figure 5. Time contributions of FT-PIR.

Table 1. Performance of PIR and FT-PIR (ms).

PIR	TPQ	TSQ	TWRG	TR	TPT	FT-PIR	TPQ	TSQ	TWRG	TR	TPT
k2t1	4.31	3.38	270.42	3.96	282.08						
	1.53%	1.20%	95.87%	1.40%	-						
k3t2	6.47	1.31	328.35	6.88	343.02	**k3t1**	2.16	5.84	345.87	9.76	363.63
	1.89%	0.38%	95.72%	2.01%	-		0.59%	1.61%	95.12%	2.68%	(+6.01%)
k4t3	13.52	5.37	278.03	5.49	302.41						
	4.47%	1.78%	91.94%	1.82%	-						
k5t4	4.21	4.06	287.03	5.51	300.81	**k5t2**	1.38	2.59	345.6	27.15	376.72
	1.40%	1.35%	95.42%	1.83%	-		0.37%	0.69%	91.74%	7.21%	(+25.24%)
k6t5	10.84	3.56	276.08	6.01	296.48						
	3.66%	1.20%	93.12%	2.03%	-						
k7t6	5.84	10.84	270.08	7.26	294.02	**k7t3**	1.88	6.66	268.44	77.08	354.06
	1.99%	3.69%	91.86%	2.47%	-		0.53%	1.88%	75.82%	21.77%	(+20.42%)
k8t7	14.24	5.36	272.29	8.66	300.56						
	4.74%	1.78%	90.60%	2.88%	-						
k9t8	14.68	6.25	298.27	10	329.2	**k9t4**	2.53	9.15	203.75	168.32	383.75
	4.46%	1.90%	90.60%	3.04%	-		0.66%	2.38%	53.09%	43.86%	(+16.57%)
k10t9	15.02	7.1	270.91	11.63	304.66						
	4.93%	2.33%	88.92%	3.82%	-						
k11t10	5.71	16.21	281.52	13.38	316.82	**k11t5**	3.95	12.28	204.28	316.09	536.61
	1.80%	5.12%	88.86%	4.22%	-		0.74%	2.29%	38.07%	58.91%	(+69.37%)

Figure 6 shows the maximum time taken to deal with queries. Throughout all the experiments, TDQmax of FT-PIR was observed to be always shorter than TDQmax of PIR, meaning that FT-PIR is always faster than PIR to get a ready group in fault-free cases. For k = 3, FT-PIR spent 35% less time than PIR in terms of the maximum time taken to deal with queries. This is simply because FT-PIR always selects the fast replicas and ignores the slow ones, while PIR has to wait for all the replicas to reply. TDQmax is quite stable for both schemes because 1) the machines have same specifications and 2) the client communicates with replicas concurrently. The standard deviation of TDQmax is always within 3% in both cases.

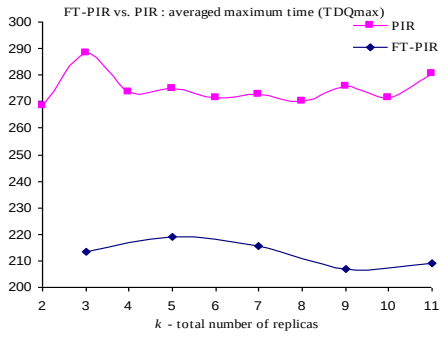

Figure 6. FT-PIR vs. PIR: averaged max time(ms).

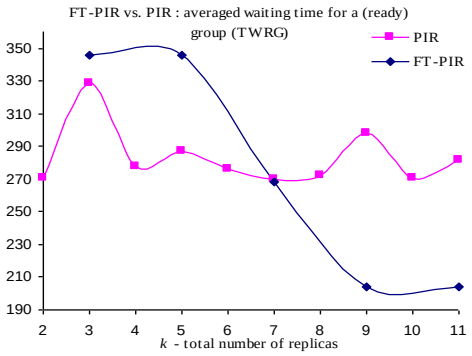

Figure 7. FT-PIR vs. PIR: averaged waiting time (ms).

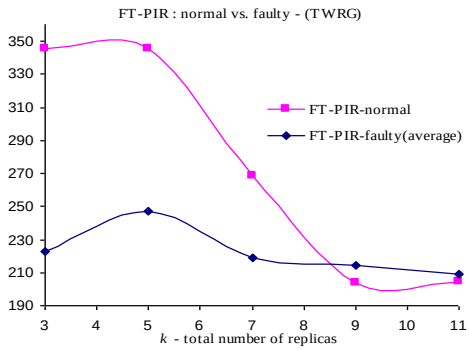

Figure 8. FT-PIR: averaged time for a ready answer group (ms).

Figure 7 shows the average time to wait for a ready group. In PIR, the standard deviation of TWRG is always within 6%. However, TWRG of FT-PIR drops dramatically as the number of replicas increases. Between three and eleven replicas, the TWRG decreases 40.94%.

B. Performance of FT-PIR in Faulty Cases

This section describes our effort in investigating the performance of FT-PIR in a faulty environment. The fault in this section is limited to fail-stop faults – replicas just stop serving any queries and deliver no response. Although FT-PIR can tolerate malicious faults, we will not discuss here the performance implication of malicious attacks. This is because simulating the behaviour of malicious attackers is a difficult task and requires a number of deliberate treatments. The related experiments will constitute the core of our further investigation.

In our experiments, replicas listen to a specific port (e.g. 5000) for incoming queries from clients. When simulating a faulty replica, we simply change this predefined port to another one (say 6000) in the client-side program. Since no programs run on port 6000, the client thread will report a failure of the specific replica. Since the maximum number of faulty replicas classifies the tests performed, there are only five categories of tests in total. We also rotated the replicas in turn to simulate faults in the hope that each replica has the same probability to fail.

Table 2 shows the performance of FT-PIR when experiencing fail-stop faults. As expected, TPQ and TSQ increase as the number of replicas increases. But the total of TPQ and TSQ is always less than 4% of TPT. TDQmax is stable for all the experiments. The standard deviation of TDQmax is identical for PIR and FT-PIR. Again, TR becomes the major contributor to TPT as the number of replicas increases.

For each category (for example, $k9t4 - f4$, $k9t4 - f3$, $k9t4 - f2$, and $k9t4 - f1$, where f is the number of actual faulty replicas), TPT does not increase as f increased. The standard deviation of TPT in the above example is within 2%. This is because TWRG and TR did not change much for the same number of f. TWRG and TR are the major factors in deciding TPT for FT-PIR.

Figure 8 shows that TWRG in faulty cases is generally less than that in the fault-free case. In fact, we only need to check less group combinations. However, as discussed before, this is also related to the speed of replicas. As expected, in Figure 9 TR is nearly the same for both cases since given same pieces of answers it should take the same time to reconstruct a result. Faults have no impact on TR in general.

Table 2. FT-PIR average performance in faulty scenarios experiments (ms): fail-stop faults.

FT-PIR	*k*3*t*1		*k*5*t*2		*k*7*t*3		*k*9*t*4		*k*11*t*5	
TPQ	1.45	0.62%	0.43	0.15%	1.5	0.49%	2.14	0.53%	2.68	0.49%
TSQ	2.25	0.96%	4.72	1.68%	8.24	2.67%	11.31	2.80%	15.14	2.75%
TWARG	223.08	95.59%	247.1	87.99%	219.12	70.93%	214.35	53.08%	209.18	37.94%
TR	6.58	2.82%	28.58	10.18%	80.07	25.92%	176.01	43.59%	324.35	58.83%
TPT	233.36	-	280.83	-	308.93	-	403.81	-	551.35	-

Figure 10 shows the performance of FT-PIR in both fault-free and faulty cases. The pattern in Figure 10 is a combination of the patterns shown in Figure 8 and 9. Initially, TWRG is the major factor in TPT. As the number of replicas increases, TR becomes a major contributor to TPT.

An important observation is that FT-PIR performs even better in the faulty cases than in the normal cases. In particular, when k = 3 (i.e. t = 1), 35.82% less TPT is observed. When k = 11 (i.e. t = 5), on average, it is only 2.75% extra time spent in the faulty cases. However, this saving decreases with an increase of k because TR (the reconstruction and verification time) becomes the major factor of TPT.

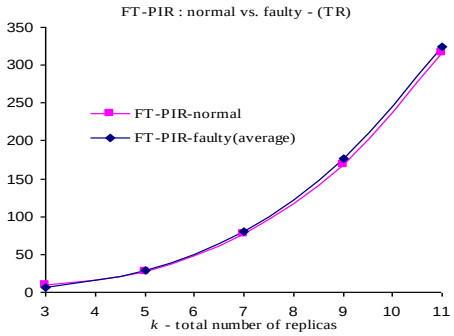

Figure 9. FT-PIR: reconstruction time (ms).

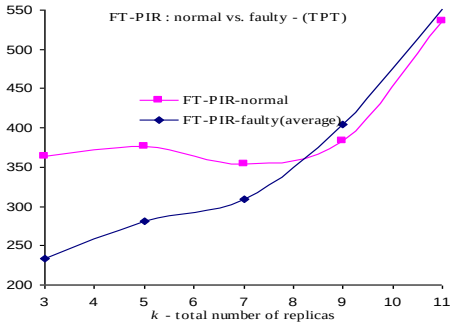

Figure 10. Performance comparisons of FT-PIR: normal cases against faulty cases (ms).

6. Related Work

There are a number of research areas related to our work: 1) PIR, 2) combination of Fault Tolerance and Security, and 3) protection of software execution in an untrusted environment. As discussed in Section 1, the PIR schemes are theory-focussed and based on the honest-but-curious model.

There are several proposals that use some form of "encrypted" data shares to achieve better security and reliability of systems. Typical examples include UC Berkeley's oceanstore [17], CMU's PASIS system [11], the Fragmentation-Redundancy-Scattering technique developed in [9, 10], and the IBM's e-Valt project [13]. MIT's Byzantine Fault Tolerance [6], the EU's MAFTIA project [5] and the COCA system [22] are other examples that address both security and fault tolerance issues of servers in asynchronous distributed systems. However, all the above proposals and systems have traditionally focussed on protecting critical information on the server side rather than the privacy of the system users.

Our work may be loosely categorised as a combination of fault tolerance and security, but with an emphasis on the protection of users' privacy from a variety of faults and guarantee the service availability for users. Unlike the above examples, our new scheme does not require vast message passing for the purpose of coordinating operations between nodes, and not rely on any unproven cryptographic premises to detect corrupted results.

Existing approaches for protecting software in an untrusted environment generally have only a limited capability to guarantee the data privacy and to tolerate malicious faults. In [20], Wang et al. proposed to use software transformation to protect software. It is expected that after transformation, an attacker may find difficult to understand the program. However, this approach could not tolerate any data tampering type attacks.

7. Conclusions and Future Work

Previous PIR research mainly demonstrated the theoretical feasibility of PIR schemes based on a simple model of binary bit strings. All the theoretical work assumed the honest-but-curious model without addressing malicious attacks. Our work shows that it is possible to enhance PIR with an ability to detect and tolerate malicious faults. A probabilistic algorithm for error detection is derived and embedded into an FT-PIR scheme. It can detect the occurrence of corrupted results, and thereby reject them with a high probability. We have also demonstrated the practical feasibility of our approach by designing and implementing the FT-PIR scheme in a real distributed database environment. The comprehensive performance analysis shows that the general performance of FT-PIR is highly acceptable and comparable with typical PIR schemes. In many cases, FT-PIR performs even better than PIR.

There are at least four possible extensions to this work in the future. First, we plan to investigate the relationship amongst different factors (e.g. the size of a database) of the system performance. Secondly, we will explore the scalability of the system by varying the number of concurrent clients. Thirdly, it would be interesting to develop schemes that protect both users' privacy and servers' privacy together in the presence of malicious faults. Finally, we plan to extend our initial work by considering a model for asynchronous distributed systems.

Acknowledgements

This work was supported by the EPSRC IBHIS project and the EPSRC/DTI e-Demand project.

References

[1] A. Ambainis, "Upper bound on the communication complexity of private information retrieval", *Proc. 24th Int'l Colloquium on Automata, Languages and Programming* (ICALP'97), LNCS, vol. 1256, Springer-Verlag, Bologna, Italy, July 1997, pp. 401-407.

[2] D. Beaver and J. Feigenbaum, "Hiding Instances in Multioracle Queries", *Proc. 7th Ann. Symposium on Theoretical Aspects of Computer Science* (STACS'90), Rouen, France, LNCS, vol. 415, Springer-Verlag, Feb.1990, pp. 37- 48.

[3] A. Beimel and Y. Ishai, "Information-Theoretic Private Information Retrieval: A Unified Construction", *Proc. 28th Int'l Colloquium on Automata, Languages and Programming* (ICALP 2001), Crete, Greece, LNCS, vol. 2076, Springer-Verlag, July 2001, pp. 912-926.

[4] C. Cachin, S. Micali, and M. Stadler, "Computationally Private Information Retrieval with Polylogarithmic Communication", *Proc. Advances in Cryptology (EUROCRYPT '99)*, Prague, Czech Republic, LNCS, vol. 1592, Springer-Verlag, May 1999, pp. 402-414.

[5] C. Cachin, "Distributing trust on the Internet", *Proc. 2001 Int'l Conf. Dependable Systems and Networks* (DSN'01), Goteborg, Sweden, July 2001, pp. 183-192.

[6] M. Castro, *Practical Byzantine Fault Tolerance*, tech. report MIT/LCS/TR-817, Laboratory for Computer Science, MIT, Cambridge, MA, USA, Jan. 2001.

[7] B. Chor, O. Goldreich, E. Kushilevitz, and M. Sudan, "Private Information Retrieval", *Proc. 36th Annual Symposium on Foundations of Computer Science* (FOCS'95), Milwaukee, Wisconsin, USA, 23-25 Oct. 1995, pp. 41-51. Journal version: *J. of the ACM*, vol. 45, no. 6, 1998, pp. 965-981.

[8] B. Chor, N. Gilboa, and M. Naor, *Private information retrieval by keywords*, tech. report TR CS0917, Dept. Computer Science, Technion, Israel, 1997.

[9] Y. Deswarte, L. Blain, and J. C. Fabre, "Intrusion Tolerance in Distributed Computing Systems", *Proc. 1991 IEEE Symposium on Research in Security and Privacy*, Oakland, California, USA, May 1991, pp. 110-121.

[10] J. M. Fray, Y. Deswarte, and D. Powell, "Intrusion-Tolerance Using Fine-Grain Fragmentation-Scattering", *Proc. 1986 IEEE Symposium on Research in Security and Privacy*, Oakland, California, USA, April 1986, pp. 194-201.

[11] G. R. Ganger, P. K. Khosla, M. Bakkaloglu, M. W. Bigrigg, G. R. Goodson, S. Oguz, V. Pandurangan, C. A. N. Soules, J. D. Strunk, J. J. Wylie, "Survivable Storage Systems", *Proc. DARPA Information Survivability Conference and Exposition*, Anaheim, CA, USA, June 2001, pp. 184-195.

[12] S. Goldwasser and M. Bellare, "Lecture Notes on Cryptography", 2001, available at: http://www.cs.ucsd.edu/users/mihir/papers/gb.html.

[13] A. Iyengar, R. Cahn, J. Garay, and C. Jutla, "Design and Implementation of a Secure Distributed Data Repository", *Proc. 14th IFIP International Information Security Conference* (SEC '98), Sep. 1998.

[14] E. Kushilevitz and R. Ostrovsky, "Replication is Not Needed: Single Database, Computationally-Private Information Retrieval", *Proc. 38th Ann. IEEE Symposium on Foundations of Computer Science* (FOCS'97), 1997, pp. 364-373.

[15] E. Kushilevitz and R. Ostrovsky, "One-way Trapdoor Permutations are Sufficient for Non-Trivial Single-Server Private Information Retrieval", *Proc. Advances in Cryptology* (EUROCRYPT 2000), LNCS, vol. 1807, Springer-Verlag, 2000, pp. 104-121.

[16] R. Lidl and H. Niederreiter, *Finite Fields*, Encyclopaedia of Mathematics and Its Applications (vol. 20), Addison-Wesley, Reading, 1983.

[17] S. Rhea, C. Wells, P. Eaton, D. Geels, B. Zhao, H. Weatherspoon, and J. Kubiatowicz, "Maintenance-Free Global Data Storage", *IEEE Internet Computing*, vol. 5, no. 5, Sept./Oct. 2001, pp 40-49.

[18] A. Shamir, "How to Share a Secret", *Communications of the ACM*, vol. 22, no. 11, 1979, pp. 612-613.

[19] M. Tompa and H. Woll, "How to Share a Secret with Cheaters", *J. of Cryptography*, vol. 1, no. 2, 1988, pp. 133-138.

[20] C. Wang, J. Davidson, J. Hill, and J. Knight, "Protection of Software-Based Survivability Mechanisms", *Proc. 2001 Int'l Conf. Dependable Systems and Networks* (DSN'01), Goteborg, Sweden, July 2001, pp. 193-202.

[21] E. Y. Yang, J. Xu and K. H. Bennett, "Private Information Retrieval in the Presence of Malicious Failures", to be published in *Proc. 26th Ann. Int'l Conf. Computer Software and Applications Conference* (COMPSAC2002), Oxford, England, Aug. 2002.

[22] L. Zhou, F. B. Schneider, and R. van Renesse, *COCA: A Secure Distributed On-line Certification Authority*, tech. report 2000-1828, Dept. Computer Science, Cornell University, Ithaca, N.Y., USA, Dec. 2000.